Cassidy

Projekt symulacji cyfrowej metodą przeglądania zdarzeń

Jakub Kruszyk

29 maja 2024

Spis treści

1 Treść zadania

Rozważmy system radiokomunikacyjny składający się z N stacji bazowych posiadających R bloków zasobów (ang. *Resource Blocks*). W losowych odstępach czasu τ (wynikającej z intensywności zgłoszeń λ) w każdej stacji bazowej pojawiają się użytkownicy. Każdy użytkownik zajmuje jeden bloków na losowy czas μ. Jeśli stacja bazowa nie ma wystarczającej liczby bloków zasobów by obsłużyć użytkownika jego zgłoszenie może być przekierowane do sąsiedniej stacji. Jeśli żadna ze stacji bazowych nie może obsłużyć zgłoszenia jest ono tracone. Intensywność zgłoszeń w systemie zmienia się cyklicznie: przez pierwsze 8 godziny intensywność zgłoszeń wynosi $\lambda/2$ przez kolejne 6 godzin - $3\lambda/4$, następnie przez 4 godziny wynosi λ, po czym spada do wartości $3\lambda/4$ na 6 godzin i cykl się powtarza. Dla stacji bazowych można ustalić próg przejścia w stan uśpienia L (wyrażony w % zajętych bloków zasobów). Stacja bazowa w stanie uśpienia pobiera moc równą 1 W, a podczas gdy jest aktywna 200 W. Zgłoszenia z uśpionej stacji są przejmowane równomiernie przez sąsiednie stacje. Podobnie jeśli w jednej z sąsiednich komórkach przekroczony zostanie próg H (wyrażony w % zajętych bloków zasobów), uśpiona komórka jest aktywowana i przejmuje połowę zgłoszeń ze stacji, w której przekroczony został próg H. Proces uśpienia i aktywacji komórki trwa 50 ms i zużywa jednorazowo 1000 W.

2 Instrukcja użytkowania

2.1 Uruchamianie programu

Program *cassidy* jest uruchamiany z wiersza poleceń w następujący sposób:

```
cassidy [Options] --duration <f64>
```

Przełącznik *duration* ustawia czas trwania symulacji wyrażonego w godzinach. Wszystkie dostępne opcje przedstawiono w tabeli 1.

Przykłady uruchamiania:

Pojedyncza iteracja, z domyślną konfiguracją przez 24 godziny czasu symulacji

```
cassidy --duration 24
```

Dziesięć iteracji, z konfiguracją wczytaną z pliku *my_cfg.toml* przez 24 godziny czasu symulacji

```
cassidy --duration 24 --iterations 10 --with-config my_cfg.toml
```

Dziesięć iteracji, z konfiguracją wczytaną z pliku *my_cfg.toml* przez 24 godziny czasu symulacji, z włączonym trybem uśpienia oraz wyświetlaniem wyników cząstkowych

```
cassidy --duration 24 --iterations 10 --with-config my_cfg.toml --enable-sleep \
        --show-partial-results
```

Pojedyncza iteracja, z konfiguracją wczytaną z pliku *my_cfg.toml* przez 24 godziny czasu symulacji, dla każdej wartości parametru zdefiniowanego w pliku *my_walk_cfg.toml*

```
cassidy --duration 24 --with-config my_cfg.toml --walk-over my_walk_cfg.toml
```

Tabela 1: Dostępne przełączniki programu *cassidy*

Opcja	Opis
--with-config <path>	Ścieżka do pliku konfiguracyjnego symulacji
--seed <u64>	Ziarno generatora liczb losowych. [domyślnie: entropia systemu]
--log	Aktywuje generowanie pliku event log
--duration <time>	Czas trwania symulacji w godzinach
--iterations <u32>	Liczba iteracji uśredniających wynik końcowy. [domyślnie: 1]
--enable-sleep	Aktywuje logikę odpowiadającą za usypianie i wybudzanie stacji bazowych
--save-default-config <path>	Domyślna konfiguracja symulacji zostanie zapisana pod ścieżką *path*
--show-partial-results	Aktywuje wyświetlanie wyników cząstkowych oraz zapisanie ich do pliku "sim_partial.run[run_no]"
--log-wave	Aktywuje generowanie pliku log w formacie binarnym
--samples <u32>	Dzielnik ilości zapisywanych próbek do binarnego pliku log [domyślnie: 1]
--walk-over <path>	Aktywuje iterowanie po zadanym parametrze z pliku pod ścieżką *path*
-h, --help	Wyświetla pomoc
-V, --version	Wyświetla wersję programu

2.2 Pliki log

Program *cassidy* może wygenerować jeden lub więcej z pięciu typów pliku log:

1. gdy przełącznik *--walk-over* NIE jest użyty, wyniki symulacji, w formacie czytelnym dla człowieka, są wyświetlane oraz zapisywane do pliku *sim_report*
2. gdy przełącznik *--walk-over* jest użyty, wyniki symulacji, w formacie *csv*, są zapisywane do pliku *sim_report*. Pierwsza linia zawiera nazwy zapisywanych sygnałów, a pierwsza kolumna zawsze zawiera wartość parametru wymienionego w pliku konfiguracyjnym
3. gdy przełącznik *--log* jest użyty, każde zdarzenie wykonane w symulacji jest zapisywane do osobnego pliku log pod ścieżką "sim.run_[run_no]_no_[iteration_no]"
4. gdy przełącznik *--log-wave* jest użyty, po każdym wykonanym zdarzeniu, zużycie zasobów oraz stan każdej stacji są zapisywane do osobnego pliku log pod ścieżką "sim_bin.run_[run_no]_no_[iteration_no]"
5. gdy przełącznik *--show-partial-results* jest użyty, wyniki pośrednie są zapisywane w formacie *csv* do pliku "sim_partial.run_[run_no]"

2.3 Pliki konfiguracyjne

2.3.1 Konfiguracja symulacji

Plik konfiguracji symulacji, w formacie *toml*, zawiera wszystkie parametry symulacji oprócz jej całkowitego czasu trwania. Poniżej znajduje się domyślna konfiguracja, którą można również uzyskać za pomocą przełącznika *--save-default-config*:

```
process_time_max = 15000 # Górny zakres czasu [ms] przetwarzania użytkownika
process_time_min = 1000  # Dolny zakres czasu [ms] przetwarzania użytkownika
lambda = 10.0            # Średnia liczba napływających użytkowników na sekundę w każdej
                         #   stacji bazowej. Ta wartość będzie przemnożona przez
                         #   współczynnik z listy lambda_coefs
resources_count = 273    # Liczba bloków zasobów w każdej stacji
sleep_threshold = 20     # Próg z zakresu <0, 100>%. Jeżeli zużycie zasobów stacji będzie
                         #   poniżej tego progu, stacja spróbuje przejść w stan uśpienia
wakeup_threshold = 80    # Próg z zakresu <0, 100>%. Jeżeli zużycie zasbów stacji będzie
                         #   powyżej tego progu, system spróbuje wybudzić jedną uśpioną stację
stations_count = 10      # Liczba stacji bazowych w systemie
active_power = 200.0     # Moc pobierana przez stację w stanie aktywnym
sleep_power = 1.0        # Moc pobierana przez stację w stanie uśpienia
wakeup_power = 1000.0    # Jednostkowa moc pobierana przez stację, która jest przełączana
                         #   ze stanu uśpienia do aktywnego i vice versa
wakeup_delay = 50        # Opóźnienie przełączania stacji ze stanu uśpienia do aktywnego
                         #   i vice versa
log_buffer = 10000       # Rozmiar bufora loggera. Obecnie to ustawienie nie ma znaczenia

[[lambda_coefs]]         # Lista par (współczynnik lambda, czas trwania)
time = 8.0               # Czas trwania, wyrażony w godzinach, tej fazy
coef = 0.5               # Współczynnik lambda tej fazy

[[lambda_coefs]]
time = 6.0               # Czas trwania fazy jest względny, dlatego ta faza będzie trwać
                         # przez 6 godzin po zakończeniu poprzedniej fazy. W tym przykładzie
                         # będzie to zakres od 8 godziny do 14 godziny czasu symulacji
coef = 0.75

[[lambda_coefs]]
time = 4.0               # Jeżeli całkowity czas trwania symulacji jest dłuższy od sumy
                         # czasów trwania wszystkich faz, cykl rozpocznie się od początku
coef = 1.0

[[lambda_coefs]]
time = 6.0
coef = 0.75
```

2.3.2 Konfiguracja iteracji

Plik konfiguracji iteracji, w formacie *toml*, zawiera nazwę parametru, po którym program będzie iterował, oraz zakres jego wartości. Program nie posiada domyślnej konfiguracji i użytkownik musi zapewnić własną.

```
var = "Lambda"  # Nazwa zmiennej po której nastąpi iteracja. Możliwe wartości to "Lambda",
                #   "SleepLow" (sleep_threshold) oraz "SleepHigh" (wakeup_threshold)
start = 10.0    # Początkowa wartość zmiennej
end = 50.0      # Końcowa wartość zmiennej
step = 5.0      # Wartość która zostanie dodana do parametru po każdej iteracji.
                # W tym przykładzie symulacja zostanie wykonana dla wartości lambda:
                # [10, 15, 20, 25, 30, 35, 40, 45, 50]
```

2.4 Skrypty pomocnicze

Aby ułatwić pracę z programem, przygotowano skrypty w języku *Python* do wizualizacji danych zapisanych w *logach* generowanych przez program. Skrypty znajdują się w folderze *scripts/* w załączonym repozytorium.

2.4.1 parse_bin_log.py

Skrypt służący do wyświetlania binarnych plików log generowanych przełącznikiem *--log-wave*.

```
Usage: python parse_bin_log.py <path_to_log> [subsampling]
```

Przykład wczytania co setnej próbki z pliku *sim.log*.

```
python parse_bin_log.py sim.log 100
```

Domyślnie zostaną wyświetlone dane tylko z pierwszej stacji. Aby wyświetlić/ ukryć dane z pozostały stacji należy klikną na odpowiednią linię na legendzie wykresu.

2.4.2 parse_csv_log.py

Skrypt służący do wyświetlania zawartości z raportu symulacji w formacie *csv* generowanego przełącznikiem *--walk-over*.

```
Usage: python parse_csv_log.py <path_to_log>]
```

2.4.3 rng_histograms.py

Skrypt do wyświetlania wyników generatorów liczb losowych czasu przetwarzania użytkownika oraz czasu do pojawienia się następnego użytkownika. Aby wygenerować dane z generatorów należy użyć poniższej komendy:

```
cargo test -- test_rng generate_lambda
```

Następnie aby wyświetlić wykresy należy użyć poniższej komendy z folderu *scripts/*:

```
python rng_histograms.py
```

3 Zasada działania programu

Przydzielona metoda symulacyjna to **metoda przeglądania zdarzeń (M1)**. W tej metodzie, w każdym obrocie pętli głównej programu, wyszukiwane jest zdarzenie (ang. *event*), którego znacznik czasowy jest najmniejszy. Następnie licznik czasu symulacji ustawiany jest na tą wartość i wykonywane jest zdarzenie. Po wykonaniu zdarzenia czasowego program sprawdza czy zostały spełnione warunki zdefiniowane dla zdarzeń warunkowych i gdy jest to wymagane, również wykonuje te zdarzenia.

Zidentyfikowane zdarzenia czasowe w zadaniu to:

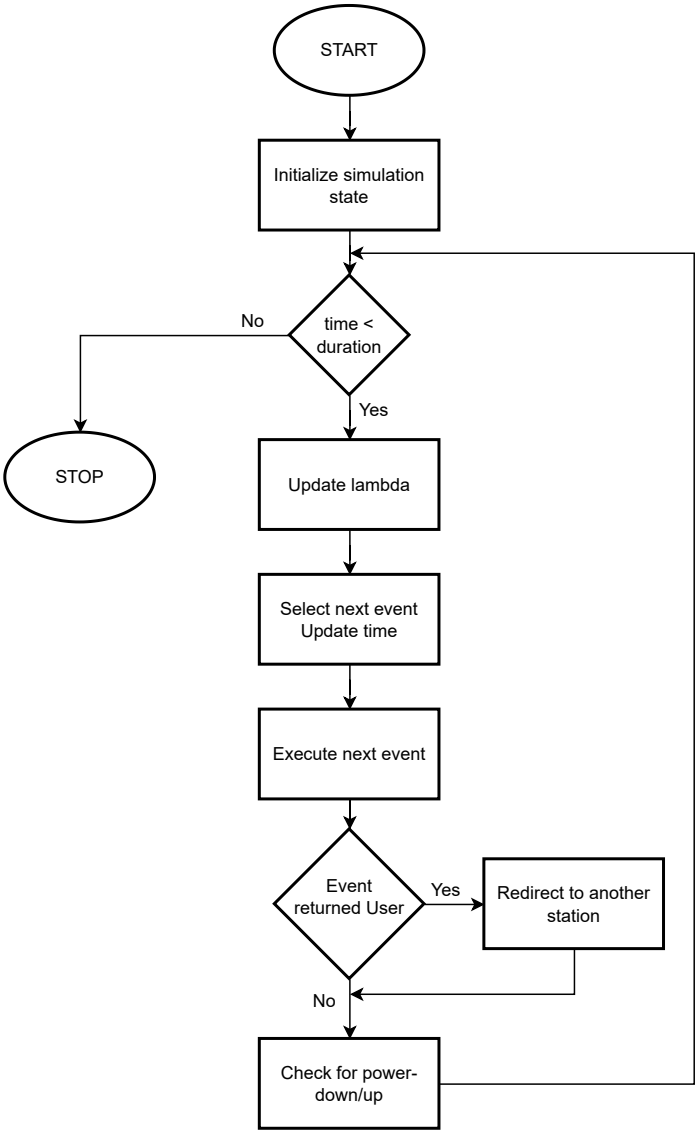
- Pojawienie się nowego użytkownika w stacji bazowej – *AddUser*
- Zakończenie obsługi użytkownika – *ReleaseUser*
- Zakończenie przejścia stacji ze stanu aktywnego do stanu uśpienia – *ShutDown*
- Zakończenie przejścia stacji ze stanu uśpienia do stanu aktywnego – *PowerUp*

Zidentyfikowane zdarzenia warunkowe w zadaniu to:

- Przekierowanie użytkownika z pełnej stacji bazowej do innej
- Rozpoczęcie przełączania stacji aktywnej w tryb uśpienia, gdy jej zużycie zasobów jest poniżej odpowiedniego progu
- Rozpoczęcie przełączania stacji uśpionej w tryb aktywny, gdy zużycie zasobów stacji aktywnej jest powyżej odpowiedniego progu

Działanie pętli głównej symulacji zostało przedstawione na rysunku 1. W pierwszej kolejności tworzona jest instancja struktury *SimState*, zawierającej stan symulacji. Następnie, tak długo jak znacznik czasu stanu symulacji jest mniejszy od całkowitego czasu trwania z konfiguracji, wykonywana jest pętla główna, na którą składają się poniższe czynności:

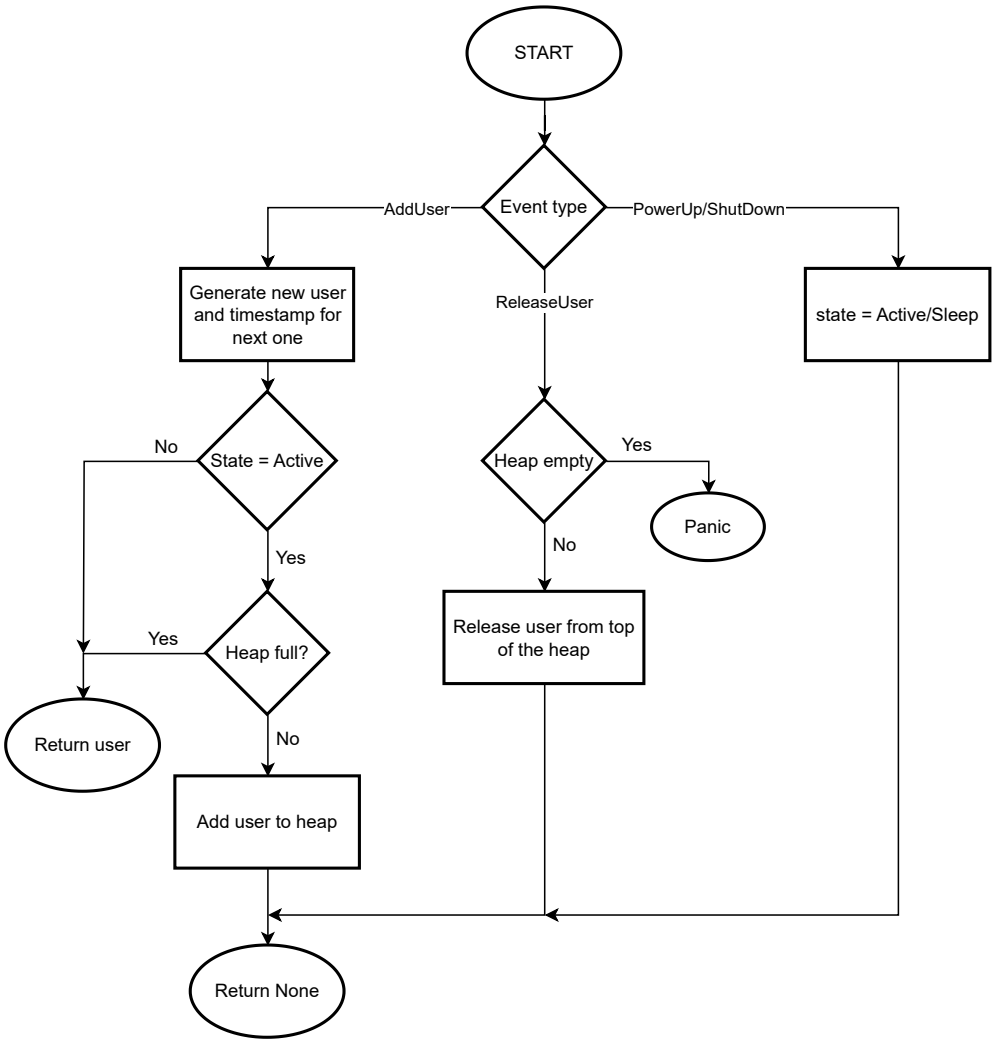
1. Aktualizacja wartości λ – następuje gdy aktualny czas symulacji jest większy od, zapisanego w stanie symulacji, czasu aktualizacji wartości λ. Wartość *lambda* z konfiguracji jest mnożona przez kolejny współczynnik z listy *lambda_coefs* oraz obliczany jest czas kolejnej aktualizacji. Obsługa tego zdarzenia została rozdzielona od obsługi zdarzeń pochodzących ze stacji bazowych, aby poprawić czytelność i wydajność programu
2. Wybór najbliższego zdarzenia – każda stacja bazowa jest odpytywana o najbliższe zdarzenie, z których wybierane jest jedno, o najmniejszym znaczniku czasu, i zapisywane w tymczasowej zmiennej. Czas symulacji jest ustawiany na wartość znacznika czasu wybranego zdarzenia
3. Wykonanie zdarzenia – Zapisane zdarzenie jest przekazywane do odpowiedniej stacji do obsługi. Pełna procedura została przedstawiona na rysunku 2. Użytkownik zostanie dodany do stacji tylko gdy jest ona aktywna oraz posiada wolne bloki zasobów. W każdym innym przypadku użytkownik zostanie zwrócony do systemu. W przypadku zdarzenia zakończenia obsługi użytkownika, jeżeli wszystkie bloki zasobów stacji są wolne, zostanie to uznane za krytyczny błąd i nastąpi przerwanie wykonywania programu
4. Obsługa przekierowania – jeżeli zdarzeniem było pojawienie się nowego użytkownika, a stacja bazowa zwróciła go do systemu w celu przekierowania, wybierana jest stacja z najmniejszym zużyciem zasobów w systemie. Jeżeli wybrana stacja ma wolny blok zasobów, użytkownik jest przypisywany do tej stacji. W przeciwnym razie użytkownik jest tracony



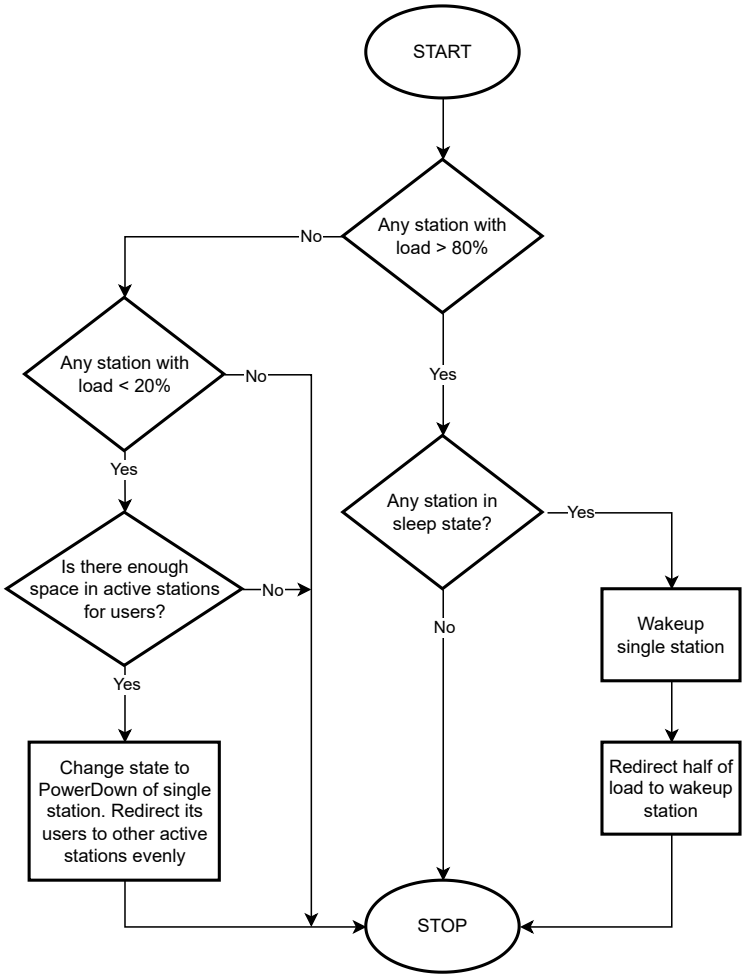
Rysunek 1: Schemat blokowy pętli głównej programu

5. Sprawdzenie możliwości uśpienia/ wybudzenia stacji – następuje gdy użyty jest przełącznik *--enable-sleep*. Jeżeli zużycie zasobów dowolnej stacji w systemie przekracza odpowiedni próg z konfiguracji, następuje próba wybudzenia jednej z uśpionych stacji i przekazania jej połowy użytkowników. W przeciwnym wypadku wyszukiwana jest stacja, której zużycie zasobów jest poniżej drugiego progu z konfiguracji. Jeżeli taka stacja została znaleziona, następuje próba przełączenia jej w stan uśpienia. Pełna procedura została pokazana na rysunku 3

Aby przyspieszyć działanie programu, wszystkie iteracje są wykonywane równolegle, za pomocą iteratorów z biblioteki *Rayon*. Z tego powodu przebiegi symulacji są niezależne względem siebie i wszystkie zasoby, niezbędne do jej wykonania, są tworzone indywidualnie dla każdego przebiegu.



Rysunek 2: Schemat blokowy pętli głównej programu



Rysunek 3: Schemat blokowy pętli głównej programu

4 Struktury

4.1 Struktury konfiguracyjne

Wszystkie struktury konfiguracyjne oraz metody bezpośrednio z nimi związane znajdują się w pliku *src/config.rs*

Struktura *Cli* zawiera pola odpowiadające wszystkim przełącznikom, których można użyć przy uruchamianiu programu. Sprawdzanie poprawności danych i konwersja na odpowiednie typy jest realizowana za pośrednictwem biblioteki *Clap*.

Struktura *Config* zawiera wszystkie pola pliku konfiguracyjnego symulacji. Wczytywanie danych jest realizowane za pomocą bibliotek *Serde* oraz *toml*. Lista współczynników λ jest reprezentowana wektorem struktur *LambdaPoint*. Każda instancja zawiera dwa pola:

- time – czas trwania danej fazy
- coef – współczynnik λ

Poprawność danych jest sprawdzana metodą *validate()* zaimplementowaną dla struktury *Config*.

Struktura *WalkOverConfig* zawiera wszystkie pola pliku konfiguracyjnego iteracji po zadanym parametrze. Wczytywanie i walidacja danych z pliku jest realizowana za pomocą bibliotek *Serde* i *toml*.

4.2 Struktura użytkownika – *User*

Struktura modelująca pojedynczego użytkownika oraz wszystkie zaimplementowane dla niej metody znajdują się w pliku *src/user.rs*. Struktura zawiera unikatowy identyfikator (*id*), czas pojawienia się użytkownika w systemie (*start*) oraz czas zakończenia obsługi (*end*), który jest sumą czasu pojawienia się i losowego czasu obsługi, losowanego z rozkładu równomiernego.

```rust
pub struct User {
    pub id: usize,
    pub start: u64,
    pub end: u64,
}

impl User {
    pub fn new(id: usize, curr_time: u64, generator: &mut StdRng, cfg: &Config) -> User {
        // convert process_time from miliseconds to microseconds
        let delay: u64 =
            generator.gen_range((cfg.process_time_min * 1000)..=(cfg.process_time_max * 1000));
        User {
            id,
            start: curr_time,
            end: curr_time + delay,
        }
    }
}
```

4.3 Struktura stacji bazowej – *BaseStation*

Struktura modelująca stację bazową i zaimplementowane na niej metody znajdują się w pliku *src/basestation.rs*. Każda instancja stacji zawiera unikatowy identyfikator (*id*), kopiec binarny reprezentujący bloki zasobów (*resources*), znacznik czasu pojawienia się kolejnego użytkownika (*next_user_add*), stan stacji (*state*) oraz liczniki zużytej mocy (*total_power*), zużytych zasobów (*total_usage*) i czasu w trybie uśpienia (*sleep_time*). Kopiec binarny został wybrany jako główny struktura danych w systemie ze względu na stałą złożoność $O(1)$ dostępu do najmniejszego elementu oraz logarytmiczną złożoność $O(logn)$ dodawania i usuwania elementów z kopca.

Znacznik czasu pojawienia się nowego użytkownika jest wyznaczany z rozkładu wykładniczego, którego gęstość prawdopodobieństwa określona jest wzorem:

$$\lambda \cdot e^{-\lambda x}$$

gdzie parametr x to liczba losowa wytworzona generatorem liczb losowych.

```
pub struct BaseStation {
    pub id: usize,
    resources: BinaryHeap<User, FnComparator<fn(&User, &User) -> Ordering>>,
    pub next_user_add: u64,
    pub state: BaseStationState,
    pub total_power: f64,
    pub total_usage: f64,
    pub sleep_time: u64,
}
```

Stan stacji bazowej jest reprezentowany jednym z wariantów typu wyliczeniowego (ang. *enum*) *BaseStationState*. Stany *PowerUp* i *PowerDown*, reprezentujące stan przełączenia pomiędzy stanami aktywnym i uśpienia, przechowują znacznik czasu zakończenia tego stanu. Zdarzenia generowane przez stację są reprezentowane przez typ wyliczeniowy *BaseStationEvent*.

```
pub enum BaseStationState {
    Active,
    Sleep,
    PowerUp(u64),   // Station is during power-up process
    PowerDown(u64), // Station is during power-down process
}

pub enum BaseStationEvent {
    ReleaseUser,
    AddUser,
    PowerUp,
    ShutDown,
}
```

Po zakończeniu symulacji, zawartości liczników są dzielone przez czas trwania symulacji, a następnie są zwracane jako instancja struktury *BaseStationResult*.

```
pub struct BaseStationResult {
    pub average_power: f64,
```

```
    pub average_usage: f64,
    pub average_sleep_time: f64,
}
```

4.4 Struktura kontenera symulacji – *SimContainer*

Struktura *SimContainer*, zdefiniowana w pliku *src/sim_container.rs*, jest abstrakcyjną strukturą modelującą system. Zawiera jedynie instancje struktur konfiguracyjnych. Stan symulacji, modelowany strukturą *SimState*, jest tworzony osobno dla każdego przebiegu symulacji. Niezależność przebiegów umożliwia proste zrównoleglenie obliczeń za pomocą biblioteki *Rayon*. Dwie najważniejsze metody zaimplementowane na strukturze *SimContainer* to:

- *simulate()* – tworzy nową instancję stanu i uruchamia jeden przebieg symulacji. Wyniki są zwracane jako instancja struktury *SimResults*, które jest zdefiniowana w pliku *src/sim_container/sim_results.rs*,
- *run()* – wywołuje równolegle metodę *simulate()* a następnie uśrednia uzyskane wyniki, które są zwracane jako pojedyncza instancja struktury *SimResults*, która zawiera uśrednione wyniki z całego systemu oraz wektor wyników z poszczególnych stacji

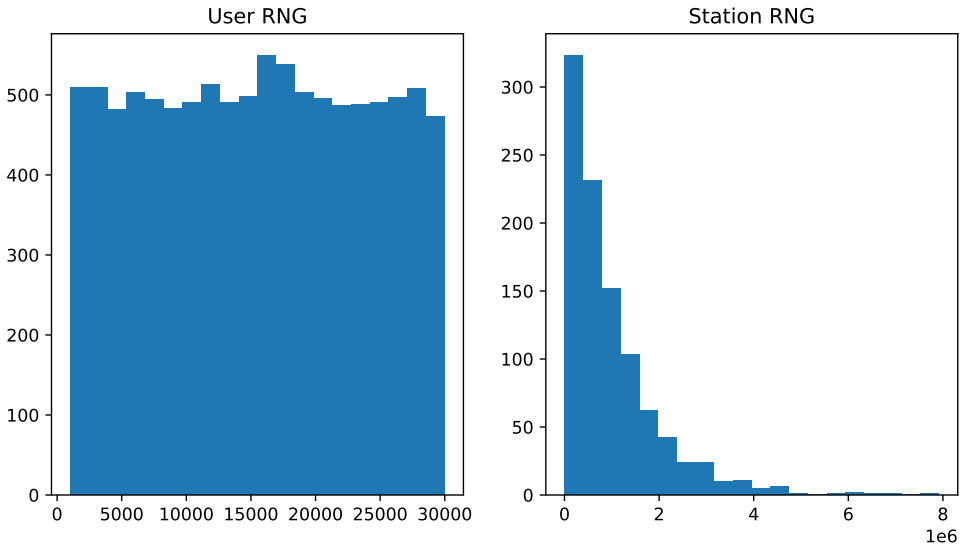
```
pub struct SimContainer {
    cli: Cli,
    cfg: Config,
}

pub struct SimState {
    pub time: u64,
    pub next_user_id: usize,
    pub lambda: f64,
    pub lambda_update_time: u64,
    pub lambda_update_idx: usize,
    pub all_users: usize,
    pub redirected_users: usize,
    pub dropped_users: usize,
}

pub struct SimResults {
    pub average_usage: f64,
    pub average_power: f64,
    pub average_drop_rate: f64,
    pub total_users: usize,
    pub dropped_users: usize,
    pub stations: Vec<BaseStationResult>,
}
```

5 Generatory liczb losowych

Na początku każdego przebiegu symulacji tworzona jest instancja generatora liczb losowych *StdRng*. Ziarno każdego generatora jest losowane ze źródeł zapewnionych przez dany system operacyjny na podstawie jego entropii. Histogramy generowanych liczb losowych z rozkładu równomiernego i wykładniczego przedstawiono na rysunku 4. Ponieważ każda iteracja symulacji musi posiadać niezależny stan wewnętrzny, aby umożliwić wykonywanie równoległe przebiegów, każda iteracja posiada osobny generator liczb losowych z indywidualnie wygenerowanym ziarnem. To rozwiązanie umożliwia także wyznaczanie parametrów statystycznych na podstawie uzyskanych wyników z wielu, krótkich przebiegów, metodą niezależnych replikacji (ang. *independent replication method*). Wymaga ona aby faza początkowa, w której system przechodzi do stanu ustalonego, była krótka względem całkowitego czasu eksperymentu. Tą własność potwierdzono eksperymentem opisanym w sekcji 7.



Rysunek 4: Histogramy reprezentujące generatory liczb losowych czasu przetwarzania użytkownika (*User RNG*) oraz czasu do pojawiania się nowego użytkownika (*Station RNG*)

6 Testowanie programu

Poszczególne funkcjonalności programu zostały przetestowane za pomocą testów jednostkowych (ang. *unit test*). Testy dotyczące danych funkcji są zawarte w module *mod test*, w tym samym pliku źródłowym oraz bazują na wbudowanej infrastrukturze testowej ekosystemu języka *Rust*. Aby uruchomić wszystkie testy wystarczy użyć komendy *cargo test* w głównym folderze projektu. Zrzut ekranu, przedstawiający pozytywny wynik wszystkich testów, zawarto na rysunku 5.

```
kkrus@PC-Kuba ~/cassidy (master)> cargo test
   Compiling cassidy v1.0.0 (/mnt/d/Studia_projekty/cassidy)
    Finished test [unoptimized + debuginfo] target(s) in 5.73s
     Running unittests src/main.rs (target/debug/deps/cassidy-cdce8c3d3d96a85b)

running 13 tests
test sim_container::test::redirect ... ok
test basestation::test::add_user_all_states ... ok
test basestation::test::add_release_user ... ok
test sim_container::test::try_wakeup ... ok
test sim_container::test::try_shutdown ... ok
test basestation::test::get_event ... ok
test basestation::test::release_user_panic - should panic ... ok
test logger::test::flush ... ok
test logger::test::logger ... ok
test basestation::test::test_add_release_order ... ok
test sim_container::test::single_sim ... ok
test basestation::test::generate_lambda ... ok
test user::test::test_rng ... ok

test result: ok. 13 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.65s

kkrus@PC-Kuba ~/cassidy (master)>
```

Rysunek 5: Przykładowy wynik komendy *cargo test*

W procesie testowania aplikacji wykorzystywano także generowane pliki *log* oraz wymienione w sekcji 2.4 skrypty napisane w języku *Python*. Umożliwiają one ręczne oraz automatyczne sprawdzanie poprawności działania pętli głównej programu. Fragment przykładowego pliku *event log* przedstawiono poniżej. Pierwsza kolumna zawiera znacznik czasu wyrażony w mikrosekundach. Następnie podany jest rodzaj zdarzenia oraz dotyczące go informacje.

```
10009189   UserAdd       Station id: 2   User id: 14, end time: 12492641 next user: 11003934
10941299   UserRelease   Station id: 0   User id: 3, end time: 10941299
11003934   UserAdd       Station id: 2   User id: 15, end time: 17544415 next user: 17034435
11161844   UserRelease   Station id: 0   User id: 5, end time: 11161844
12097745   UserAdd       Station id: 0   User id: 16, end time: 36047089 next user: 13170839
12492641   UserRelease   Station id: 2   User id: 14, end time: 12492641
13100375   UserRelease   Station id: 2   User id: 1, end time: 13100375
13170839   UserAdd       Station id: 0   User id: 17, end time: 34192635 next user: 14459706
14299594   UserRelease   Station id: 0   User id: 8, end time: 14299594
```

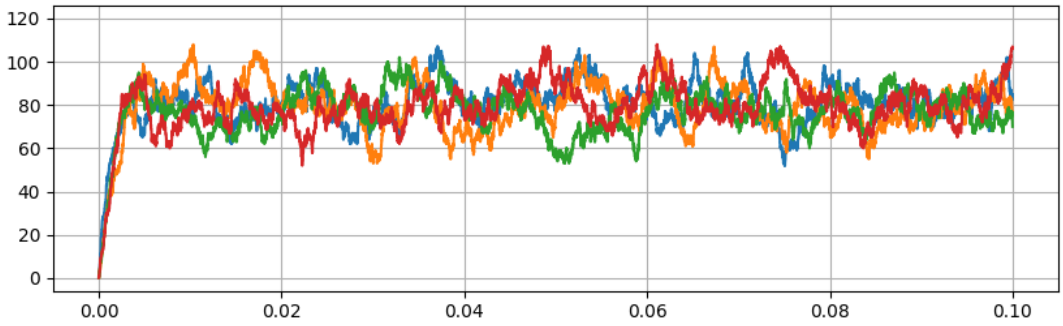
7 Wyniki symulacji

7.1 Wyznaczenie długości fazy początkowej

Długość fazy początkowej wyznaczono za pomocą binarnego pliku *log* zawierającego ilość zajętych bloków zasobów każdej stacji w funkcji czasu. Uzyskane wyniki dla pięciu różnych wartości parametru λ przedstawiono w tabeli 2. Na podstawie uzyskanych wyników stwierdzono skracanie czasu trwania fazy początkowej wraz ze wzrostem wartości parametru λ oraz uznano jej wpływ na wyniki trwania eksperymentu za nieznaczące. Przykładowy wykres, na podstawie którego wyznaczono wyniki przedstawiono na rysunku 6. Analizując wyniki należy mieć na uwadze subiektywność uzyskanych wyników, ze względu na nieprecyzyjną definicję końca fazy początkowej.

Tabela 2: Wyznaczony czas fazy początkowe dla różnych wartości λ

Lambda	20	25	30	35	40
Czas fazy początkowej [s]	38	36	30	27	20



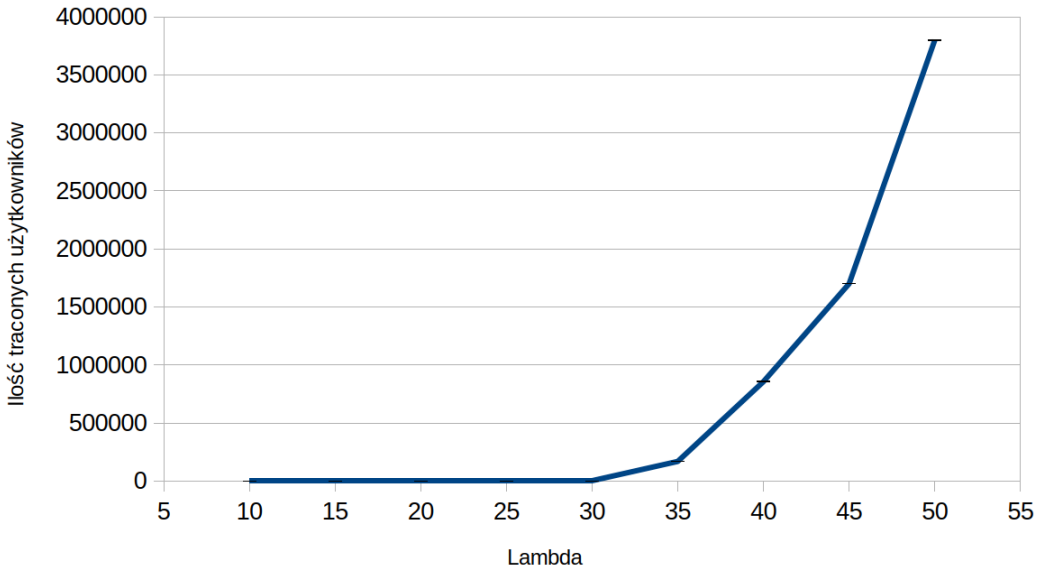
Rysunek 6: Wykres ilości zajętych bloków zasobów stacji bazowych w funkcji czasu wyrażonego w godzinach. Parametr λ podczas symulacji był równy 20

7.2 Wyznaczenie granicznej wartości λ, dla której nie stracono żadnego użytkownika

Aby wyznaczyć wartość graniczną, uruchomiono symulację dla wartości λ z zakresu $< 10; 50 >$ użytkowników na sekundę, z krokiem 5. Każda próbka została uśredniona z 10 iteracji. W trakcie symulacji logika odpowiadająca za usypianie stacji była wyłączona. Wyniki symulacji, wraz z obliczonymi przedziałami ufności, przedstawiono w tabeli 3 oraz na rysunku 7. Ostatnią wyznaczoną wartością λ, dla której nie stracono żadnego użytkownika, to 30 użytkowników na sekundę.

Tabela 3: Wyniki pomiaru ilości traconych użytkowników w systemie w funkcji wartości parametru λ

lambda	Ilość straconych użytkowników	Przedział ufności
10	0	0
15	0	0
20	0	0
25	0	0
30	0	0
35	168720	1152
40	856671	1686
45	1700934	1755
50	3799727	1605



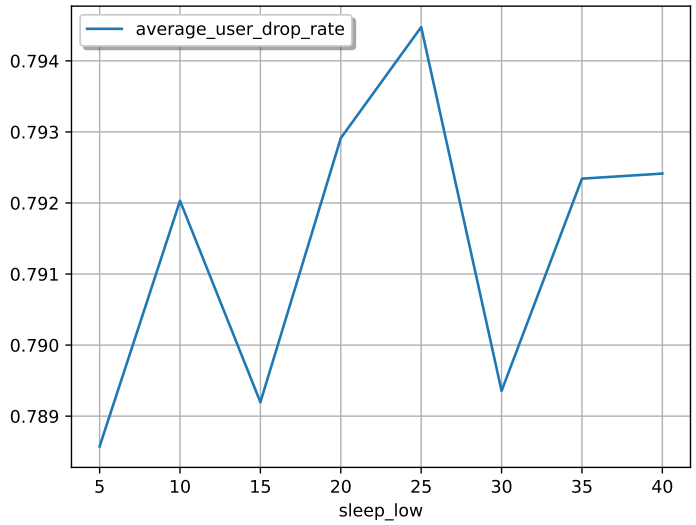
Rysunek 7: Wykres liczby traconych użytkowników w funkcji wartości parametru λ

7.3 Wyznaczenie granicznej wartości progu L, dla której straty użytkowników są poniżej 5%

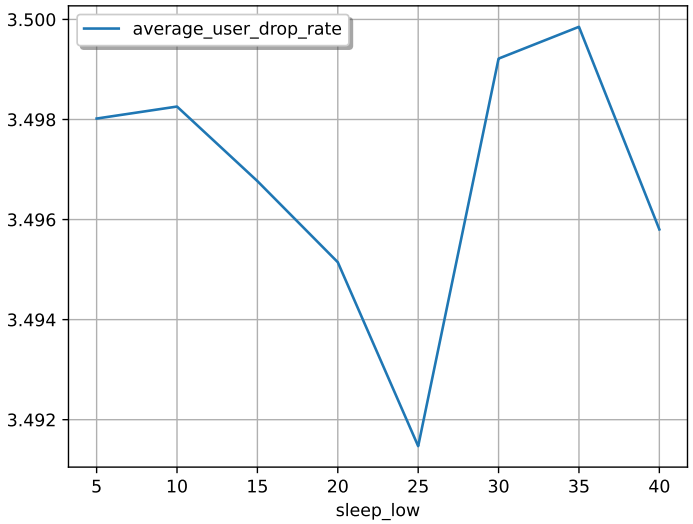
Aby wyznaczyć wartość graniczną, uruchomiono symulację dla wartości progu L z zakresu $< 5; 40 >$ %, z krokiem 5. Każda próbka została uśredniona z 10 iteracji. W trakcie symulacji parametr λ wynosił 35. Wynik symulacji przedstawiono na rysunku 8. Z wyników symulacji wynika, że ilość traconych użytkowników w systemie nie zależy od wartości progu L. Eksperyment powtórzono dla wartości $\lambda = 40$ oraz 50, których wyniki przedstawiono na rysunkach 9 i 10. W każdym przypadku różnice pomiędzy uzyskanymi wynikami są pomijalnie małe. Wynika to najprawdopodobniej z zastosowanego algorytmu usypiania

i wybudzania stacji bazowych, który został zaprojektowany tak aby maksymalnie zminimalizować ilość traconych użytkowników w systemie. Analizując wykres zużycia zasobów stacji w czasie, przedstawionego na rysunku 11, można zauważyć, że badana wartość utrzymuje się na w przybliżeniu stałym poziomie zależnym od współczynnika λ w danej fazie. Aby wygenerować straty użytkowników przekraczające 5%, parametr λ powinien być duży, na przykład z zakresu $< 50; 70 >$ użytkowników na sekundę, aby zapełnić bloki zasobów we wszystkich stacjach. Zakres ten został wyznaczony osobnymi symulacjami, z wyłączoną logiką odpowiadająca za usypianie i wybudzanie stacji. Duża wartość parametru λ jest także wymagana aby w systemie pojawiła się wystarczająca liczba użytkowników, aby przepełnić aktywne stacje, zanim zostanie wybudzona stacja uśpiona. Natomiast aby próg L, wskazujący poniżej jakiej wartości stacja bazowa może przejść w stan uśpienia, miał wpływ na przebieg symulacji, wartość parametru λ musi być niska, aby zużycie bloków zasobów stacji spadło poniżej progu L. Jest to sprzeczne z postawionymi wcześniej założeniami i dlatego taka sytuacja nie zachodzi w systemie, dla podanych w treści zadania parametrów. Dlatego do dalszej części tego zadania, opisanego w sekcji 7.4 przyjęto następujące parametry symulacji:

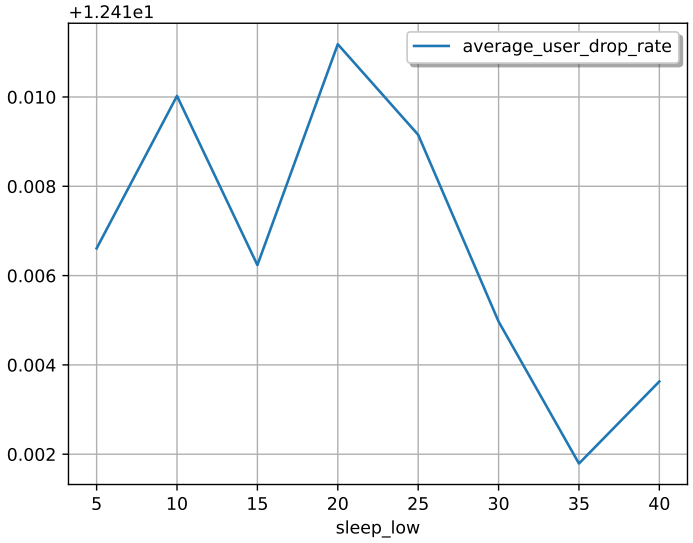
- Lambda = 43 użytkowników na sekundę
- L = 20%



Rysunek 8: Wykres procentu traconych użytkowników w funkcji wartości parametru L. W trakcie symulacji parametr λ wynosił 35.



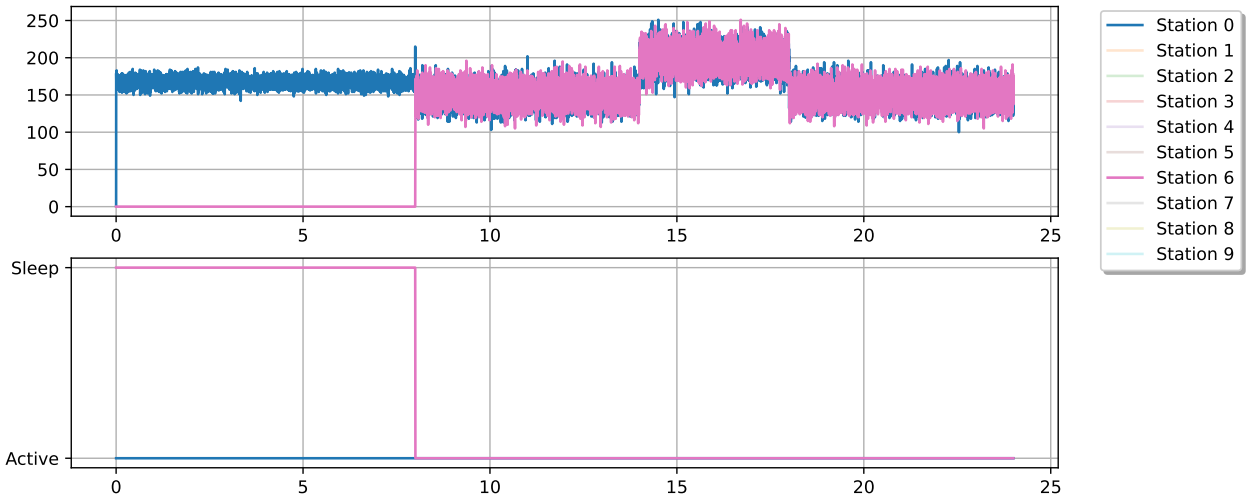
Rysunek 9: Wykres procentu traconych użytkowników w funkcji wartości parametru L. W trakcie symulacji parametr λ wynosił 40.



Rysunek 10: Wykres procentu traconych użytkowników w funkcji wartości parametru L. W trakcie symulacji parametr λ wynosił 50.

7.4 Wyznaczenie uśrednionych parametrów systemu

Uśrednione parametry systemu wyznaczono symulacją uśrednioną z 20 iteracji, gdzie parametr λ wynosił 43 a próg L był równy 20%. Wyniki symulacji oraz raport końcowy z programu przedstawiono odpowiednio w tabeli 4 oraz na rysunku 12. Ponieważ ilość użytkowników napływających do systemu była większa od pojemności stacji bazowych, wszystkie uśpione stacje zostały wybudzone w początkowej fazie symulacji, stąd średnia moc pobierana przez stacje wynosi 200W a średni czas uśpienia to 0%. Parametry



Rysunek 11: Wykres zużycia zasobów stacji bazowych nr 0 i 6 w funkcji czasu symulacji

statystyczne zostały wyznaczone na podstawie wyników pośrednich, uzyskanych z użyciem przełącznika *--show-partial-results*, za pomocą wbudowanych funkcji programu *Excel*.

```
=================== Average simulation results ===================
Simulation results:
- processed users: 26315377
- dropped users: 1286804
- average resource usage: 84.88 %
- average power consumption: 199.99 W
- average user drop rate: 4.89 %

Stations results:
id  | average power [W] | average usage [%] | average sleep time [%]
----+-------------------+-------------------+-----------------------
 0  |      200.00       |       84.91       |         0.00
 1  |      200.00       |       84.91       |         0.00
 2  |      200.00       |       84.90       |         0.00
 3  |      199.99       |       84.90       |         0.00
 4  |      199.99       |       84.89       |         0.01
 5  |      199.99       |       84.88       |         0.01
 6  |      199.99       |       84.88       |         0.01
 7  |      199.98       |       84.87       |         0.01
 8  |      199.98       |       84.86       |         0.01
 9  |      199.98       |       84.83       |         0.01
```

Rysunek 12: Raport wynikowy symulacji dla parametrów $\lambda = 43$ oraz L = 20, uśredniony z 20 iteracji

Tabela 4: Wyniki mierzonych parametrów systemu w każdej iteracji

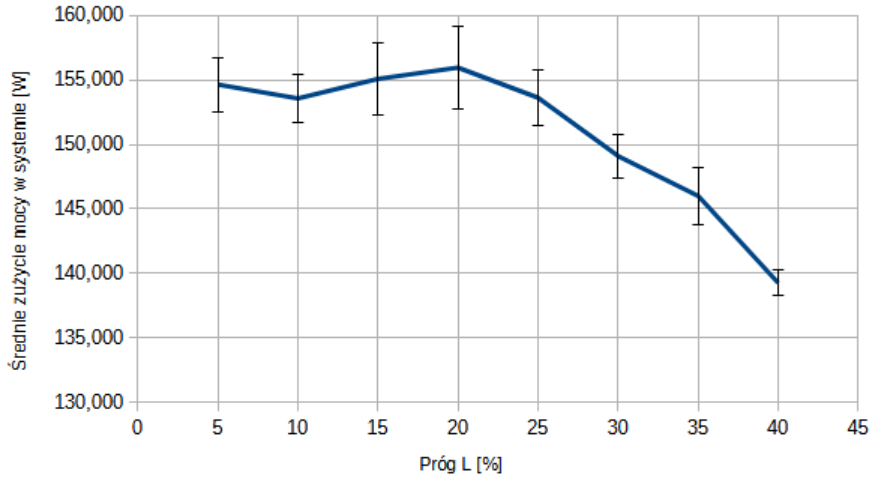
Iteracja	Średnie zużycie zasobów [%]	Średni pobór mocy [W]	Średni procent traconych użytkowników	Średni czas uśpienia
0	84,891	200	4,899	0
1	84,877	200	4,897	0
2	84,899	200	4,902	0
3	84,897	200	4,894	0
4	84,863	200	4,890	0
5	84,897	200	4,896	0
6	84,863	200	4,905	0
7	84,864	200	4,876	0
8	84,890	200	4,909	0
9	84,884	200	4,891	0
10	84,898	200	4,890	0
11	84,890	200	4,868	0
12	84,873	200	4,896	0
13	84,892	200	4,892	0
14	84,896	200	4,880	0
15	84,874	200	4,896	0
16	84,893	200	4,895	0
17	84,880	200	4,906	0
18	84,901	200	4,896	0
19	84,891	200	4,891	0
Wartość średnia	84,886	200	4,893	0
Odchylenie standardowe	0,0126	0	0,010	0
Przedział ufności	0,006	-	0,004	-

7.5 Wyznaczenie średniego zużycia energii w funkcji progu L

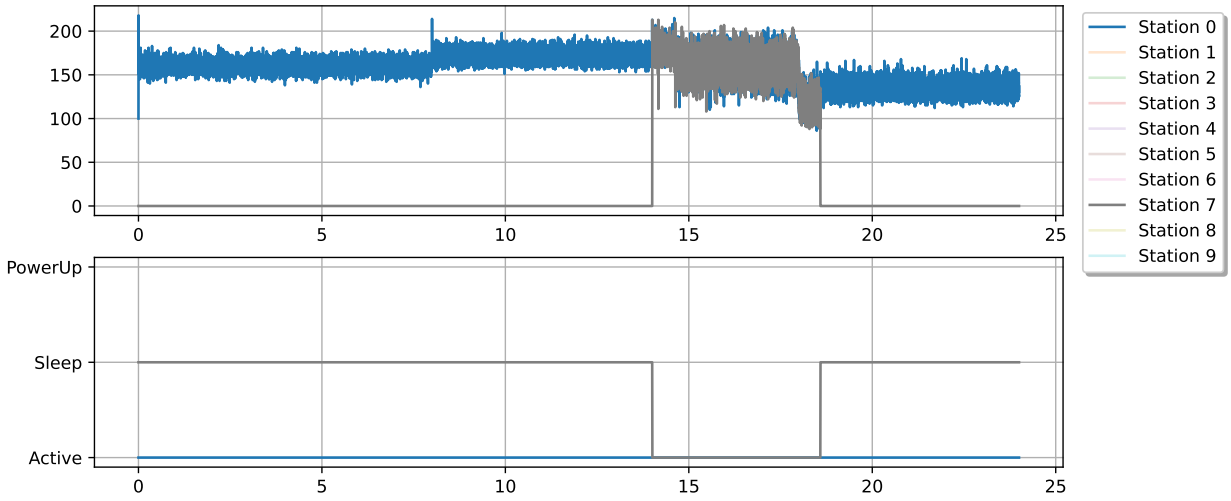
Symulację wykonano dla wartości progu L z zakresu $< 5 : 40 >$ z krokiem równym 5, dla wartości parametru $\lambda = 20$. Wyniki symulacji przedstawiono w tabeli 5 oraz na rysunku 13. Każda próbka wykresu została uśredniona z 10 iteracji. Z analizy uzyskanego wykresu wynika że wraz ze wzrostem progu uśpienia L, dla niskich wartości λ, średni pobór mocy przez stacje bazowe maleje. Na rysunku 14 przedstawiono wykres zużycia zasobów stacji 0 i 7 w funkcji czasu. Stacja numer 7 została aktywowany po rozpoczęciu fazy największego ruchu (gdy współczynnik $\lambda\_coef = 1$) oraz została wyłączona gdy generowany ruch zmalał. Dzięki temu średni pobór mocy w systemie zmalał, względem symulacji gdy próg uśpienia miał niższą wartość lub gdy logika odpowiedzialna za usypianie stacji była wyłączona.

Tabela 5: Wyniki pomiaru średniego zużycia mocy w funkcji wartości parametru L

Próg L [%]	Średnie zużycie mocy [W]	Przedział ufności
5	154,633	2,060
10	153,566	1,833
15	155,074	2,759
20	155,949	3,213
25	153,625	2,151
30	149,099	1,698
35	146,011	2,196
40	139,263	0,989



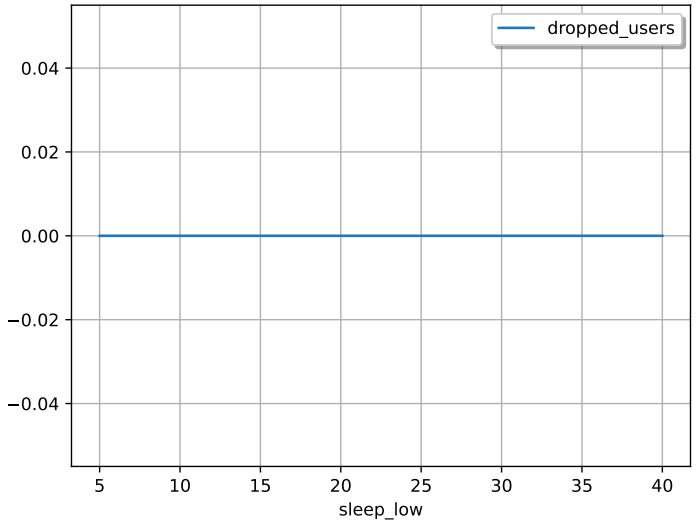
Rysunek 13: Wykres średniego poboru mocy przez stacje bazowe w funkcji wartości progu L. W trakcie symulacji parametr λ był równy 20



Rysunek 14: Wykres średniego poboru mocy przez stacje bazowe w funkcji wartości progu L

7.6 Wyznaczenie średniej ilości traconych użytkowników w funkcji progu L

Z przyczyn wymienionych w sekcji 7.3, liczba traconych użytkowników nie zależy od wartości progu L oraz, ponieważ w trakcie symulacji parametr λ był równy 20, jest równa zero.



Rysunek 15: Wykres średniego ilości traconych użytkowników w funkcji wartości progu L